Advanced practical Programming for Scientists

Thorsten Koch
Zuse Institute Berlin
TU Berlin

WS2014/15
set \( I := \{ 1 \ldots \text{size} \}; \)

var \( x[I][I] \) binary;

subto c1: # rows half / half
forall \( <i> \in I \) do
  sum \( <j> \in I: x[i][j] = \text{size} / 2; \)

subto c2: # cols half / half
forall \( <i> \in I \) do
  sum \( <j> \in I: x[j][i] = \text{size} / 2; \)

subto c1: # never three equal color in a row
forall \( <i> \in I \) do
  forall \( <j> \in I \setminus \{1, 2\} \) do
    1 \leq x[i][j - 2] + x[i][j - 1] + x[i][j] \leq 2;

subto c1: # never three equal color in a col
forall \( <i> \in I \) do
  forall \( <j> \in I \setminus \{1, 2\} \) do
    1 \leq x[j - 2][i] + x[j - 1][i] + x[j][i] \leq 2;
How does it work?

0001 & 1110 + 1 = 1111 = 0001
0010 & 1101 + 1 = 1110 = 0010
0011 & 1100 + 1 = 1101 = 0001
0100 & 1011 + 1 = 1100 = 0100
0101 & 1010 + 1 = 1011 = 0001
...

How does this work?

```c
if (updatemask & 0xffff0000)
    colidx += 16;
if (updatemask & 0xff00ff00)
    colidx += 8;
if (updatemask & 0xf0f0f0f0)
    colidx += 4;
if (updatemask & 0xcccccccccc)
    colidx += 2;
if (updatemask & 0xaaaaaaaaaa)
    colidx += 1;
```
```c
if (updatemask & 11111111111111110000000000000000b)
colidx += 16;
if (updatemask & 11111111000000001111111110000000b)
colidx += 8;
if (updatemask & 11110000111100001111000011110000b)
colidx += 4;
if (updatemask & 11001100110011001100110011001100b)
colidx += 2;
if (updatemask & 10101010101010101010101010101010b)
colidx += 1;
```
Black-Box-Testing is a method of software testing that is applied without any knowledge of the inner working of the system to be tested.

It is limited to functionality related tests, i.e. the test cases are derived from the requirements not from the implementation of the test object.

The program is treated as a black box. Only properties that are visible to the outside are tested.

**Example:** Running test inputs into a system/function and checking whether the outcome is as expected.

Does it do the right thing?

**Disadvantage:** Completeness impossible
White-Box-Testing
(also known as clear box testing, glass box testing, transparent box testing, and structural testing) tests the internal structures or workings, as opposed to its functionality. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs.

Examples:
• Line coverage: Execution of all source lines
• Instruction (node) coverage: Execution of all instructions
• Branch (edge) coverage: All branches of the control flow are executed
• Path coverage: Execution of all paths through a module

Does it work in the intended way?

Disadvantage: intended and correct are not the same thing.
Test Driven Development

Do the tests first.

How to do this in our case?

How to do this in more complicated cases?

How to do this in research, where you are the first to answer a question?
How to grade you?

I think I have to measure something like the amount of learning? / understanding? / knowledge? you possess after the course? / have gained from the course?

Measurement is defined as:

*The process of assigning symbols, usually numbers, to represent an attribute of the entity of interest, by rule.*

Maybe I grade you by the, should we call it quality?, of the code you submit to the exercises.

How to we do this?
A **software metric** is a measure of some property of a piece of software or its specifications. Since quantitative measurements are essential in all sciences, there is a continuous effort by computer science practitioners and theoreticians to bring similar approaches to software development.

The goal is obtaining **objective, reproducible and quantifiable measurements**, which may have numerous valuable applications in schedule and budget planning, cost estimation, quality assurance testing, software debugging, software performance optimization, and optimal personnel task assignments.

What properties of a piece of software might have? (that we can measure by rule)
Cyclomatic Complexity

- **Cyclomatic complexity** is a software metric (measurement).
- It was developed by Thomas J. McCabe, Sr. in 1976 and is used to indicate the complexity of a program.
- It is a quantitative measure of the complexity of programming instructions.
- It directly measures the number of linearly independent paths through a program's source code.
- Cyclomatic complexity is computed using the control flow graph of the program: the nodes of the graph correspond to indivisible groups of commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command. Cyclomatic complexity may also be applied to individual functions, modules, methods or classes within a program.
- One testing strategy, called basis path testing by McCabe who first proposed it, is to test each linearly independent path through the program; in this case, the number of test cases will equal the cyclomatic complexity of the program.
Example

const char* weekdayname(int num)
{
    switch(num) {
        case 1: return "Montag";
        case 2: return "Dienstag";
        case 3: return "Mittwoch";
        case 4: return "Donnerstag";
        case 5: return "Freitag";
        case 6: return "Samstag";
        case 7: return "Sonntag";
    }
    return "(unbekannter Wochentag)";
}

CC = 8

advancing

const char* weekdayname(int num)
{
    const char* tage[] =
    { "Montag", "Dienstag", "Mittwoch",
      "Donnerstag", "Freitag",
      "Samstag", "Sonntag"
    }
    int len = sizeof(tage);
    if ((num >= 1) && (num <= len))
        return tage[num - 1];
    return "(unbekannter Wochentag)";
}

CC = 3
The MIPLIB is a diverse collection of challenging real-world mixed integer programming (MIP) instances from various academic and industrial applications suited for benchmarking and testing of MIP solution algorithms.

The Benchmark set consist only of instances that could be solved in 2010 within two hours on a high-end personal computer by at least two MIP solvers.

MIPLIB 5 still contains 2 instances from Version 2, 7 instances from Version 3, and 17 instances from Version 4.
easy can be solved within an hour on a contemporary pc with a state-of-the-art solver

hard are solvable but take a longer time or require specialized algorithms

open, i.e. not solved instances for which the optimal solution is not known
Development of MIPLIB 2010

- Easy
- Hard
- Open
Some MIP Solvers then and now

<table>
<thead>
<tr>
<th>Solver</th>
<th>Version 2011</th>
<th>Version 2012</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM CPLEX</td>
<td>12.2</td>
<td>12.4</td>
<td><a href="http://www.cplex.com">www.cplex.com</a></td>
</tr>
<tr>
<td>Gurobi</td>
<td>4.5 beta0</td>
<td>5.0</td>
<td><a href="http://www.gurobi.com">www.gurobi.com</a></td>
</tr>
<tr>
<td>SCIP/spx</td>
<td>2.0.1</td>
<td>3.0.0</td>
<td>scip.zib.de</td>
</tr>
<tr>
<td>CBC</td>
<td>2.6.4</td>
<td>2.7.7</td>
<td>projects.coin-or.org/Cbc</td>
</tr>
</tbody>
</table>

Further solvers

<table>
<thead>
<tr>
<th>Solver</th>
<th>Version</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glpk</td>
<td>4.47</td>
<td><a href="http://www.gnu.org/software/glpk/glpk.html">www.gnu.org/software/glpk/glpk.html</a></td>
</tr>
<tr>
<td>lp_solve</td>
<td>5.5</td>
<td>lpsolve.sourceforge.net</td>
</tr>
<tr>
<td>Lindo</td>
<td>7.1</td>
<td><a href="http://www.lindo.com">www.lindo.com</a></td>
</tr>
<tr>
<td>Minto</td>
<td>3.1</td>
<td>coral.ie.lehigh.edu/~minto</td>
</tr>
<tr>
<td>Symphony</td>
<td>5.4</td>
<td>projects.coin-or.org/SYMPHONY</td>
</tr>
</tbody>
</table>
Performance relative to number of threads

Numbers courtesy of K. Fujisawa
Performance variation due to permutation

Fig. 3: Solution times for 100 permutations
Total number of B&B nodes processed by Gurobi 4.5.0

- pigeon-10
- glass4
- noswot
- gmu-35-40
- timtab1

## B&B nodes

*Computational Integer Programming*  Thorsten Koch  Zuse Institute Berlin (ZIB)
How to compare MIP solver performance?

There are two components to it:

- **Ability to solve an instance** (within some timeframe)
- **Time to solve** (heuristics see also primal integral)

You have always to see both numbers!

If two solvers are compared there are two possibilities:

1. Both can solve all instances.
2. There are instances only one solver can solve
3. There are instances both cannot solve
2012 Performance on MIPLIB 2010 / 1 thread

Instances sorted by solution time

Seconds

<table>
<thead>
<tr>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>3500</td>
</tr>
<tr>
<td>3000</td>
</tr>
<tr>
<td>2500</td>
</tr>
<tr>
<td>2000</td>
</tr>
<tr>
<td>1500</td>
</tr>
<tr>
<td>1000</td>
</tr>
<tr>
<td>500</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBC/1</td>
<td>Yellow</td>
</tr>
<tr>
<td>CPLEX/1</td>
<td>Blue</td>
</tr>
<tr>
<td>Gurobi/1</td>
<td>Green</td>
</tr>
<tr>
<td>SCIP/spx/1</td>
<td>Black</td>
</tr>
<tr>
<td>XPRESS/1</td>
<td>Purple</td>
</tr>
<tr>
<td>Best/1</td>
<td>Red</td>
</tr>
</tbody>
</table>
2012 Performance on MIPLIB 2010 / 12 threads

Instances sorted by solution time

- 2012-Best/1
- 2012-CBC/12
- 2012-CPLEX/12
- 2012-Gurobi/12
- 2012-XPRESS/12
- 2012-Best/12
2012 Speedup 1:12 threads CBC/SCIP

Instances sorted by solution time

Seconds

CBC/1

SCIP/spx/1

CBC/12

SCIP/spx/12

T. Koch, G. Gamrath, H. Mittelmann: Any Progress One Year after MIPLIB 2010 ?
### Progress in Numbers

<table>
<thead>
<tr>
<th></th>
<th>2011</th>
<th>2012</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best/1 geom. mean time [s]</td>
<td>105</td>
<td>81</td>
<td>23%</td>
</tr>
<tr>
<td>Best/12 geom. mean time [s]</td>
<td>34</td>
<td>28</td>
<td>17%</td>
</tr>
<tr>
<td>Best/1 not optimal [#]</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Best/12 not optimal [#]</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Speedup 1 to 12 [×]</td>
<td>3.1</td>
<td>2.8</td>
<td></td>
</tr>
<tr>
<td>Max(C,G,X)/Min(C,G,X) 1 thr.</td>
<td>600</td>
<td>259</td>
<td></td>
</tr>
<tr>
<td>Max(C,G,X)/Min(C,G,X) 12 thr.</td>
<td>1138</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>Best/1 geom. mean nodes [#]</td>
<td>2108</td>
<td>1958</td>
<td>8%</td>
</tr>
<tr>
<td>Best/12 geom. mean nodes [#]</td>
<td>2570</td>
<td>2406</td>
<td>7%</td>
</tr>
<tr>
<td>Max(C,G,X)/Min(C,G,X) 1 thr.</td>
<td>40,804</td>
<td>32,600</td>
<td></td>
</tr>
<tr>
<td>Max(C,G,X)/Min(C,G,X) 12 thr.</td>
<td>10,499</td>
<td>887,849</td>
<td></td>
</tr>
</tbody>
</table>

Single solvers have speed-ups up to 42%
2012 Parallel speedup Best 1/12 per instance
Speedup Best/12 2011:2012 per instance
1998 Benchmark Set

Instances sorted by solution time

Seconds

- SIP 1.1 [CPLEX 5.0 for LP] UltraSparc 167 MHz
- CPLEX 5.0 UltraSparc 167 MHz
- SIP 1.1 [CPLEX 5.0.1 for LP] X5260 3333 MHz
- SCIP 2.1.1 [CPLEX 12.4 for LP] X5260 3333 MHz
- CPLEX 12.4 X5260 3333 MHz
Read  Ada for the C++ or Java Developer